FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY
TELECOMMUNICATION DEPARTMENT

**(Msc- 2ND Semester)**


# Techniques and Algorithm for Signal Processing

## Final Project Report

**Submitted By:**

**Miss. Roshani Thaware (317616)**
**Mr. Chijioke Nnanna Erua (317608)**

**Wrasaw, Poland**

## Abstract:

**The corona virus COVID-19 pandemic is causing a worldwide emergency in healthcare. This virus mainly spreads through droplets which emerge from a person infected with the virus and poses a risk to others who are not infected. The risk of transmission is highest in public places where a large amount of people are gathered together. One of the best methods to stay safe from getting infectedby this virus is by wearing a face mask in open territories as indicated by the World Health Organization (WHO). Reports indicate that wearing face masks while one is in these public places clearly reduces the risk of transmission of the virus. In this project, we propose a method which employs OpenCV and some machine learming algorithms to detect face masks on people. A bounding box drawn over the face of the person describes weather the person is wearing a mask or not. Since our goal is to identify whether the person on image/video stream is wearing a face mask or not, In our proposed system we made use of a live video stream and finally in the output gave an indication writing whether the individual detected is wearing mask.**

## Introduction:

In recent decades, facial recognition has become the object of research worldwide. In addition, with the advancement of technology and the rapid development of artificial intelligence, very significant advances have been made. For this reason, public and private companies use facial recognition systems to identify and control the access of people in airports, schools, offices, and other places. On the other hand, with the spread of the COVID-19 pandemic, government entities have established several biosafety regulations to limit infections. Among them is the mandatory use of face masks in public places, as they have been shown to be effective in protecting users and those around them.

Facial recognition has been widely used for security and other law enforcement purposes. However, since COVID-19 pandemic, many people around the world had to wear face masks. This model introduces a neural network system, which can be trained to identify people's facial features while half of their faces are covered by face masks. One large Face mask detection dataset was first used to train the model, while the original much smaller Face mask detector dataset was used to adapt and finetune this model that was previously generated. During the training and testing phases, network structures, and various parameters were adjusted to achieve the best accuracy results for the actual small dataset.

Our proposed algorithm for the detection of face mask through video feed from different sources like webcam, video file or an IP camera was majorly achieved using several python libraries, mathematical computations, machine learning techniques and an algorithm called Haar Cascade. Haar Cascade is an Object Detection Algorithm which is used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola

and Jones. Majorly series of image processing analysis are carried out on images collected during period of observation. At this point it is important to note that images are basically represented as a matrix containing colour values for each pixel. Our adapted model was able to achieve a 97.1% accuracy.

## System Development:

We propose to design a system that is capable of identifying a person's face, even if it is with or without a mask. For the system to work properly, it is necessary to use two databases: the first is for classifier training and consists of a large number of images of people who wear a face mask and others who do not. The second is used for training the facial recognition system, and here there are people with and without the biosafety material (face mask). The input data are obtained either from an image, or a life video stream using videocapture, with the aim of having a better precision and robustness. This project is divided into three major stages, which are described below:

We will implement the algorithm governing the mode of operation of the face mask detection system in four phases. Each stage with the help of their respective algorithm will provide us with necessary data collection and image transformations to aid in the detection of the required object which is a face mask. The implementation is composed of 4 main stages:

1. Test-Image Loading and processing
2. Face detection on an Image
3. Data Collection (with Mask and Without mask) using Videocapture
4. Model training/Testing/Accuracy score

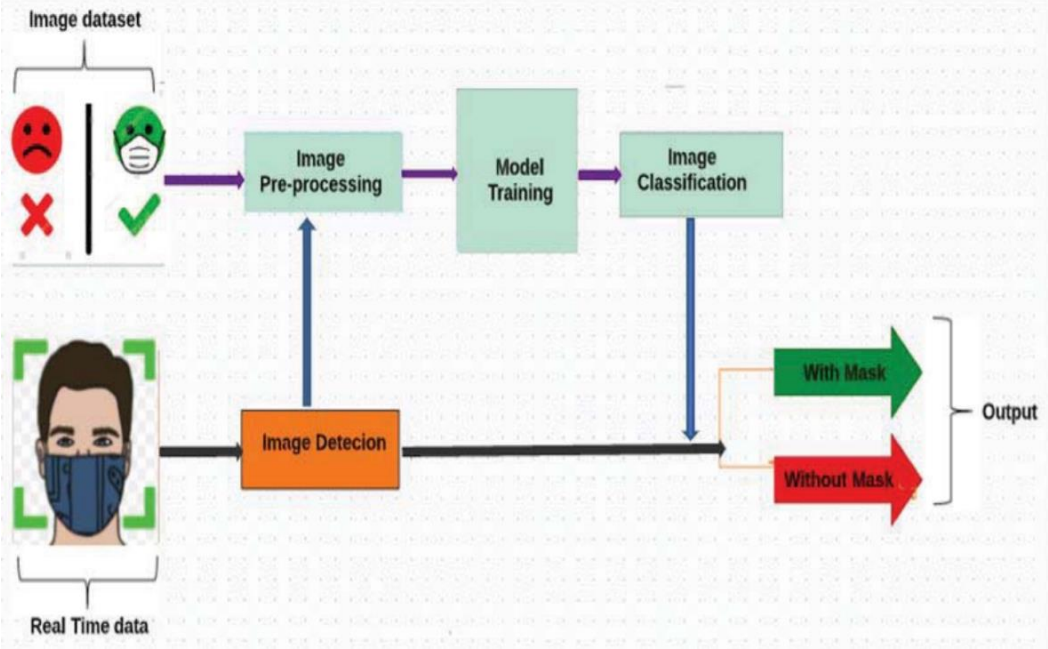


*Fig: Processing stages in real-time face mask detection*

The four stages are implemented in Python with the help of libraries, documentations and other functions. Image loading and processing was achieved using the following libraries:

## Installing the needed libraries:

- *Open Cv –*

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis.

It played a major role in the implementation of the first stage of our project. We utilized it's operational capacity in loading and processing images in such a way that these images were converted into array structure for further image analysis.

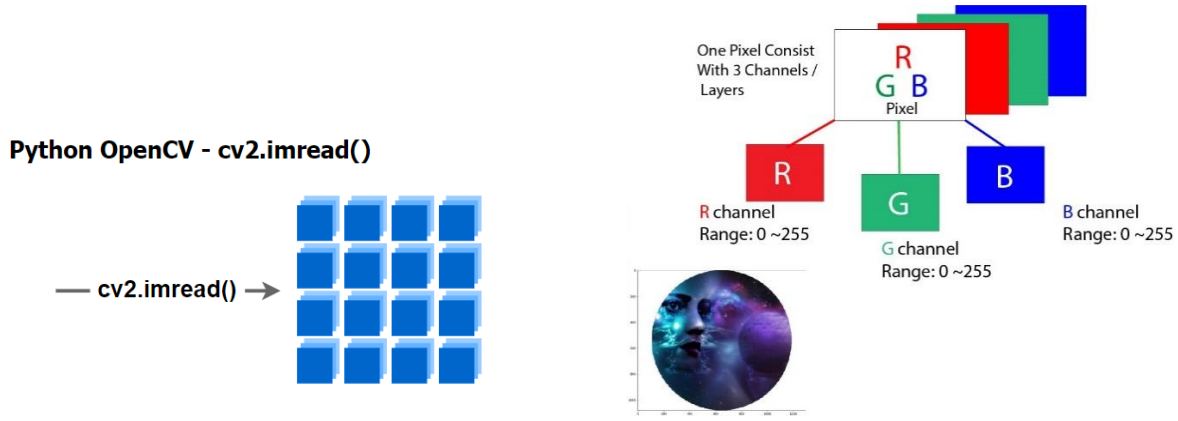


*Fig: Array representation of images and their color values.*

```
In [23]: # installing openCV
         !pip install opencv-python

         Requirement already satisfied: opencv-python in c:\users\shree\anaconda3\lib\site-packages (4.5.4.60)
         Requirement already satisfied: numpy>=1.19.3 in c:\users\shree\anaconda3\lib\site-packages (from opencv-python) (1.20.3)

In [24]: #importing libraries
         import cv2
         import matplotlib.pyplot as plt
         import numpy as np
```

*Fig: Image showing the first two lines in the first segment of the source code*

**From the above chunk of code we have the following libraries dicussed below**

- Matplotlib:

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

- Numerical python (numpy)

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. NumPy is a Python package. It stands for 'Numerical Python'

## Test Image Loading and Analysis

### Cv2.imread(img.jpg)

cv2. imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix. Therefore in the 3rd line of our source code in the figure below, the image read function is used for loading the test image whose format is a jpg type.

```
In [67]: #Reading the test image
         img = cv2.imread('img.jpg') #te

In [68]: #extracting the dimension of the image
         img.shape

Out[68]: (450, 720, 3)

In [69]: #first pixel representation in test image using array
         img[0]

Out[69]: array([[241, 237, 236],
                [241, 237, 236],
                [241, 237, 236],
                ...,
                [235, 232, 228],
                [236, 233, 229],
                [236, 233, 229]], dtype=uint8)
```

*Fig: Image showing the 3rd to 5th lines in the first segment of the source code*

By **using numpy.array()** function we can take an image as the argument and converts it to NumPy array. When these arrays are created, they determine the RGB value an image has. Digital images use some colour model to create a broad range of colours. The adopted model is the RGB (Red, Green, Blue) model which are mixed together to

form other colours. In the RGB model, the primary colours are red, green, and blue – thus the name of the model. The amount of the primary colour added is represented as an integer in the closed range [0, 255]. Therefore, there are 256 discrete amounts of each primary color that can be added to produce another color. The number of discrete amounts of each color, 256, corresponds to the number of bits used to hold the color channel value, which is eight (28=256). With this RGB values, we can move on in deploying the necessary mathematical computation and algorithm needed to achieve our goal of face mask detection.

```
In [81]: #plotting the test image in graphical form
         plt.imshow(img)

Out[81]: <matplotlib.image.AxesImage at 0x2e343249340>
```


*Fig: Loading of the test image*

**Plot.imshow(img)** function is generally used for creating static, animated, and interactive visualizations in their array format. To visualize how the generated array looks like in an image format we make use of the above said function. From the above figure we can see how the so-called function is applied in plotting the test image in a graphical form. This pattern of plotting removes the RGB color in the test image.

```
In [82]: while True:
             cv2.imshow('result', img)
             #27 - ASCII number of Escape
             if cv2.waitKey(2) == 27: #wait
                 break
         cv2.destroyAllWindows()

         #the wait key waits for an operation ifinetly or for a given delay.
         #Therefore, when the escape key is pressed, it waits for exactly 2milli-seconds before executing the next line of code.
```

*Fig: Figure showing the number of millisececond the image window has to to be available*

Looking at the Chunk of code above, we can see a ‘while loop’ which is running infinitely until it is broken by an ‘if statement’. ‘27’ represents the ASCII number of the escape key. Therefore, the loop will be broken and the running window will be closed when we press the escape key on our keyboard. The function ‘waitkey’ waits for a key event infinitely or for delay in milliseconds, when it is positive. Since the OS has a minimum time between switching threads (i.e tasks running on the computer background as can be seen on the task manager) the function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is running on your computer at that time before it switches to perform another process on the OS. As seen from the code we made use of 2 milli-seconds.

## Detecting the faces from the test image using Haarcascade:

Face detection on an image is achieved by making use of the Viola jones Algorithm. Detection of distinct objects can be easily done by humans, but a computer needs precise instructions and constraints to complete this task. To make the task possible, Viola–Jones developed an algorithm that requires full view frontal and upright face positioning. Thus in order forthis algorithm to detect a face, the entire face must point towards the camera and should not be tilted to either side.

The Algorithm by Viola and Jones is exceptionally powerful. It possesses an extremely high rate of detection and very few false-detection rate (1 in 105). Due to its detection rate, it is fast enough to be used for events occuring in real-time with speeds ranging in 2.5/sec. For practical applications involving frame rate of 2/sec.The only drawback is that this algorithm is just applicable in detecting faces and not for recognising faces. The processing stages of this algorithmis divided into 4 phases:

1. **Haar Feature Processing Segment**:

Simply defined, Haar features, are attributes taken out fromaparticular area of a sample based of the different contracsting shades of color that exixts in this sample then represented in concrete binary formto reduce the complexity of further analysis that would be carried oujt with it. Every face has some striking feature with another for example, the eyes section has a darker shade than the section where the upper cheeks is located, the nose is lighter and brighter than the eye segment and equally the segment of the forehead is more bright than the eyes and on.

2. **Integral Image Creation segment**:

As a result of irrugularities of different faces, listed below are 4 basic features used for establishing the areas of similar resemblance in different human faces.

- The eye section which has darker shade of than the upper-section of the cheeks
- The nose bridge/upper segment of the nose which is lighter than the eyes segment.
- Positional orientation and size of the features (eyes mouth and nose).

- Pixel intensities and their different gradient values.

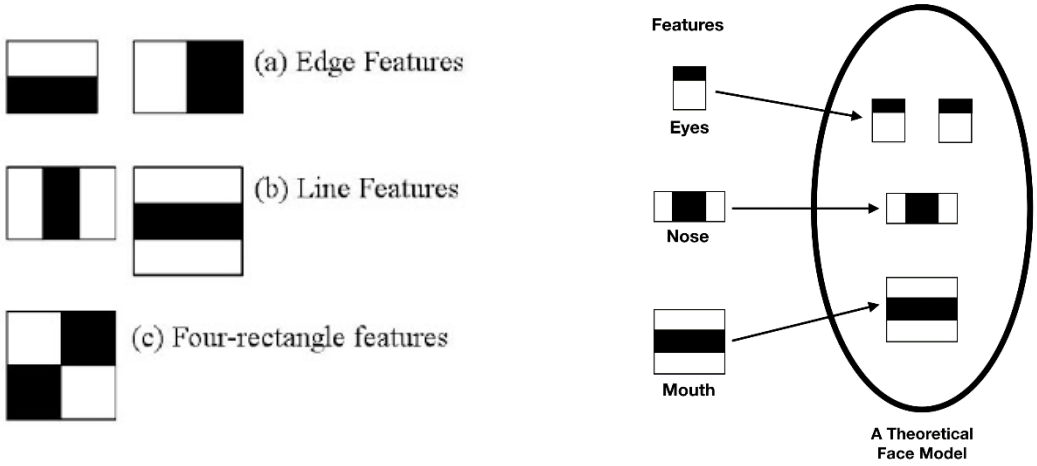


*Fig : Haar cascade features*

Viola Jones algorithm makes use of a double - rectangle feature selection with definition : summation of all black pixels – summation of all white pixels.

### 3. AdaBoost Phase:

This is a very decisive and exceedingly important phase of the face detection carried out by the Viola Jones Algorithm. AdaBoost creates a certain number of weak classifiers based on the features which will have a substantial amount of error but will then ultimately end up creating a linear combination of all the weak classifiers to create one strong classifier which is robust and definite.

The hypothesis is defined as**: $C(x)= \theta(\Sigma ht(x) +b)$**

where ht(x) has a value of either 1 or -1 depending on the classification of being a face or a non-face. This ht(x) is the weak classifier which is multiplied by a weight matrix θ. 'b' is a constant term.

The advantage of this approach is that the training error quickly converges to 0 and is extremely simple to implement as well with high real- time accuracy.

### 4. Segment of Cascading Classification:

Now once the classifier is established, a number of classifiers are arranged one after the other to minimize the effective number of false positive values gotten so as to improve the accuracy of the model. Different categories of classifiers are selected, based on the number of properties they decide to contrast between different samples and then are arranged one after one another. For example If we select 3 different classifiers which filter the outputs one after the other with error of 50%, 40%, & 20%. The effective net error rate of the whole model is going to be 2%. Thus, cascading helps to reduce the number of error resultsgenerated by the model.

```
In [72]: #uploadind the harrcascade classifier file used for detecting faces

         harr_data = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

In [76]: #detecting the haar features in the two faces from test image and returning it as an array.

         harr_data.detectMultiScale(img)

Out[76]: array([[164, 125, 208, 208],
                [327, 145, 172, 172]])

In [74]: #cv2.rectangle(img,(x,y),(w,h),(b,g,r),border_thickness)

In [75]: #drawing the rectangle on detected face from test image
         while True:
             faces = harr_data.detectMultiScale(img)
             for x,y,w,h in faces:
                 cv2.rectangle(img,(x,y),(x+w, y+h), (255,0,255), 1)
             cv2.imshow('result', img)
             #27 - ASCII of Escape
             if cv2.waitKey(2) == 27:
                 break
         cv2.destroyAllWindows()
```

*Fig: Face detection section of the source code*

From the figure above, the first line of the code shows the importation of the haarcascade file which is to be used for the whole face detection process. The next line in the above code goes further to detect the various haar features which are used in classifying an object as a face and then the array produced as this is carried out - [164, 125, 208, 208] and [327, 145, 172, 172] represents the dimension of the triangle on the faces of the first and second girl on the test image respectively. The last line produces the output which is shown below:

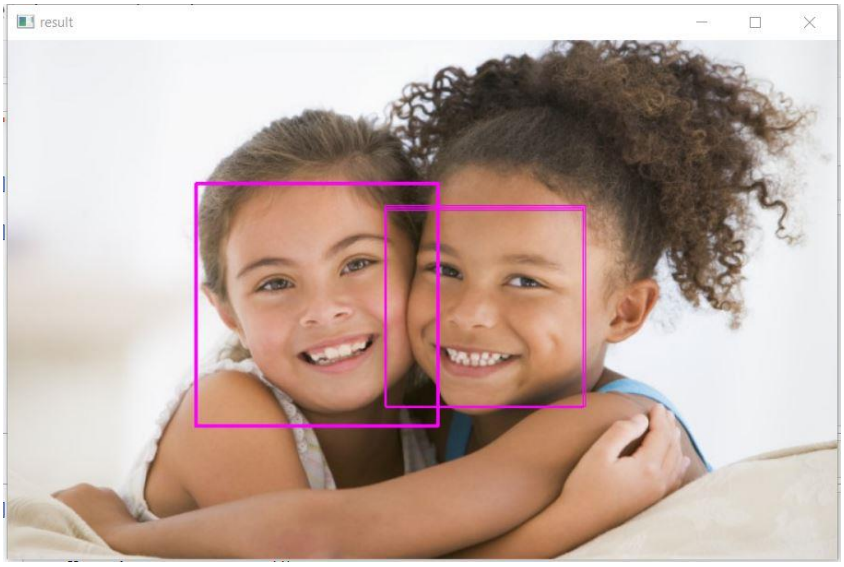


*Fig: Detecting the faces from the test image*

## Data Collection (with Mask and without mask) using Videocapture

**Cv2.Videocapture()** is the function used for collecting data. Since the harcascade algorithm has been used in detecting a face in a preloaded image its effectiveness has been ascertained, the next thing is to collect data containing images of a human face when it is masked up and not masked up. The images are captured with the help of videocam and are separated into two datasets. The first set contains images of the human face without mask and the second dataset contains images of the humn face with mask. The images captured by the system's webcam requires pre-processing before it is applied to the next stage of the algorithm.

```
In [64]: #collecting the dataset for with and without Mask from live camera
capture = cv2.VideoCapture(0)
data =[]
while True:
    flag, img = capture.read()
    if flag:
        faces = harr_data.detectMultiScale(img)
        for x,y,w,h in faces:
            cv2.rectangle(img,(x,y),(x+w, y+h), (111, 0, 111), 1)
            face = img[y:y+h, x:x+w, :]
            #resizing all detected face
            face= cv2.resize(face, (50,50))
            print(len(data))
            if len(data) < 400: #Collecting 400 images for the dataset for with and without mask
                data.append(face)
        cv2.imshow('result', img)
        #27 - ASCII of Escape
        if cv2.waitKey(2) == 27 or len(data) >=400:
            break
capture.release()
cv2.destroyAllWindows()
```

```
In [23]: #saving the collected data for without mask in .npy file format
np.save('without_mask.npy', data)
```

```
In [ ]: #saving the collected data for with mask in .npy file format
np.save('with_mask.npy', data)
```

*Fig. Collecting dataset for with and without mask from live stream camera and saving it in .npy format.*

*Fig: Few collected images for the dataset (with and without face mask)*

```
In [ ]: #resizing all two dataset images with same shape (2 X 2 dimension)
        with_mask = with_mask.reshape(200, 50 * 50 * 3)
        without_mask = without_mask.reshape(200, 50 * 50 * 3)

In [37]: #checking the new shape of with mask dataset (2 X 2 dimension)
         with_mask.shape

Out[37]: (200, 7500)

In [38]: #checking the new shape of without mask dataset (2 X 2 dimension)
         without_mask.shape

Out[38]: (200, 7500)
```

*Fig. Reshaping of the collected dataset*

The captured images are resized to the shape of (50x50) pixels size to maintain uniformity of the input images to the architecture. Then, the images are normalized and after normalization, the value of a pixel resides in the range from 0 to 1. Normalization helped the learning algorithm to learn faster and captured necessary features from the images.

## Model training/Testing/Accuracy score

In this phase of our algorithm, we intoduce the machine learning concept which will help our model to learn from experience (datasets we collected from the prior stage) so that it can make better predictions and give better outputs. Machine learning is basically divided into two categories namely supervised and unsupervised learning. Since we will employ the supervised learning technique on our model, we are going to encounter some difficulties one of which is accurate classification of data. To solve this problem, we are going to use an algorithm called support vector machine algorithm which is one of the tools in the python library called scikit learn library commonly referred to as sk-learn.

Support Vector Machine (SVM) is a supervised machine learning algorithm which is both applied in solving problems related to regressison and classification as we earlier pointed out. However, its more applied when challenges of classification is the bane of the problem. In the SVM algorithm, each data item is plotted in such a way that they represent a point in n-dimensional space (n represents the number of features the data item under analysis possesses) and the vector value of each property equates to the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well as shown in the figure below:

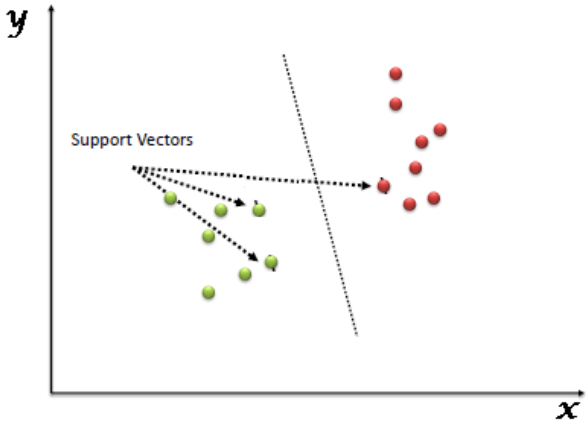


*Fig. Support Vectors and the coordinates of individual observation.*

From the above figure we can see that Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

We also make use of a function called train_test_split which is basically used to divide our data into the training data and the testing data sets. It was divided in such a way that 25% of our data will be used for testing while 75% of the data will be used for machine learning and since the shape of our train data was not suitable, we applied a technique called dimensionality reduction to reduce the number of columns of out train data because it is exceedingly high. Once this reduction is done we equally made use of the sk-learn library once again to test the accuracy of our model and check whether we need to alter some more parameters inorder to attain a maximum efficiency rate.

```
In [44]: #svm - Support Vector Machine
         #SVC - Support Vector Classification
         from sklearn.svm import SVC
         from sklearn.metrics import accuracy_score
```

```
In [45]: #splitting of data into trainning and testing datasets

         from sklearn.model_selection import train_test_split
```

```
In [46]: #75% of the data will be used for training
         #25% of the data will be used for testing

         x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size = 0.25)
```

```
In [47]: #training datasize
         x_train.shape
```

```
Out[47]: (300, 7500)
```

```
In [48]: # demensionality deduction with principle component analysis
         from sklearn.decomposition import PCA
```

```
In [49]: #converting the data from 7500 dimensional to 3 dimensional data
         pca = PCA(n_components = 3)
         x_train = pca.fit_transform(x_train)
```

In [50]:
```
x_train[0]
```

Out[50]:
```
array([-1726.08782196,  1287.19659157,  1919.76863029])
```

In [51]:
```
#current shape of data
x_train.shape
```

Out[51]:
```
(300, 3)
```

In [52]:
```
#shuffuling of data
x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size = 0.30)
```

In [53]:
```
svm = SVC()
svm.fit(x_train, y_train)
```

Out[53]:
```
SVC()
```

In [54]:
```
y_pred = svm.predict(x_test)
```

In [55]:
```
#checking the accuracy
accuracy_score(y_test, y_pred)
```

In [56]:
```
#face mask detection
haar_data =cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
capture = cv2.VideoCapture(0)
data = []
font = cv2.FONT_HERSHEY_COMPLEX
while True:
    flag, img = capture.read()
    if flag:
        faces = haar_data.detectMultiScale(img)
        for x,y,w,h in faces:
            cv2.rectangle(img, (x,y), (x+w, y+h), (255,0,255), 4)
            face = img[y:y+h, x:x+w, :]
            face= cv2.resize(face, (50,50))
            face= face.reshape(1, -1)
            pred = svm.predict(face)[0]
            n = names[int(pred)]
            cv2.putText(img, n, (x,y), font, 1, (244,250,250), 2)
            print(n)
        cv2.imshow('result', img)
        #27-ASCII of Escape
        if cv2.waitKey(2) == 27:
            break

capture.release()
cv2.destroyAllWindows()
```

## RESULT ANALYSIS

By preserving a reasonable proportion of different classes, the dataset is partitioned into training and testing set. The dataset comprises of 800 samples in total where 75% is used in training phase and 25% is used in testing phase. The developed architecture is trained by reshuffling the training and a tseting dataset continuously which inturn helps to reduce overfitting on the training data. Overfitting generally occurs when a model learns the unwanted patterns of the training samples. Hence, training accuracy increases but test accuracy decreases. Fig. shows the prediction of our model when an image from a live stream is feed into it. The trained model showed approximately 98% accuracy.

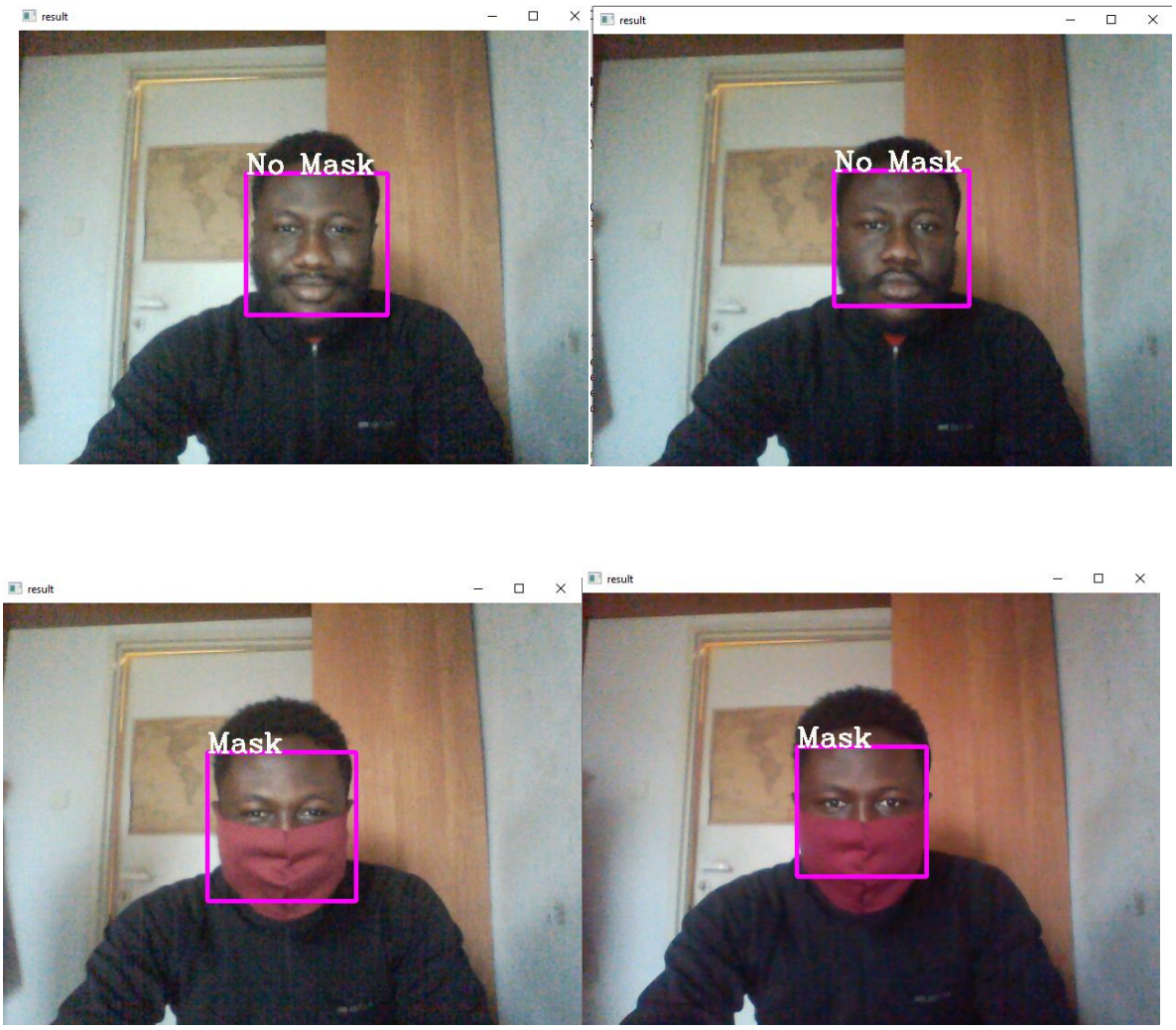


*Fig: Results of detecting the face with and without mask.*

## REFERENCES


[1] Z. Pei, H. Xu, Y. Zhang, M. Guo, and Y.-H. Yang, “Face Recognition via Deep Learning Using Data Augmentation Based on Orthogonal Experiments,” Electronics, vol. 8, pp. 1-16, 2019, doi:10.3390/electronics8101088.

[2] I. Adjabi, A. Ouahabi, A. Benzaoui, and A. Taleb-Ahmed, “Past, Present, and Future of Face Recognition: A Review,” Electronics, vol. 9, no. 8, 2020, doi:10.3390/electronics9081188.

[3] S. J. Elias, S. M. Hatim, N. A. Hassan, L. M. Abd Latif, R. B. Ahmad, M. Y. Darus, et al., “Face recognition attendance system using Local Binary Pattern (LBP),” Bulletin of Electrical Engineering and Informatics, vol. 8, no. 1, pp. 239-245, 2019, doi:10.11591/eei.v8i1.1439.

[4] S. M. Bah and F. Ming, “An improved face recognition algorithm and its application in attendance management system,” Array, vol. 5, 2020, doi:10.1016/j.array.2019.100014.

[5] O. Sanli and B. Ilgen, “Face detection and recognition for automatic attendance system,” in Proceedings of SAI Intelligent Systems Conference, pp. 237-245, 2018, doi:10.1007/978-3-030-01054-6_17.

[6] L. N. Soni, A. Datar, and S. Datar, “Implementation of Viola-Jones Algorithm based approach for human face detection,” Int. J. Curr. Eng. Technol., vol. 7, no. 5, pp. 1819-1823, 2017.

[7] F. F. Alkhali and B. K. Oleiwi, “Smart E-Attendance System Utilizing Eigenfaces Algorithm,” Iraqi Journal of Computers, Communication and Control & Systems Engineering, vol. 18, no. 1, pp. 56-63, 2018.

[8] S. Sawhney, K. Kacker, S. Jain, S. N. Singh, and R. Garg, “Real-Time Smart Attendance System using Face Recognition Techniques,” in 2019 9th International Conference on Cloud Computing, Data Science & Engineering
(Confluence), pp. 522-525, 2019, doi: 10.1109/CONFLUENCE.2019.8776934.

[9] K. Jin, X. Xie, F. Wang, X. Gao, and G. Shi, “Real-Time Face Attendance Marking System in Non-cooperative Environments,” in Proceedings of the 2018 the 2nd International Conference on Video and Image Processing, pp. 29-34, 2018, doi:10.1145/3301506.3301546.

[10] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multitask cascaded convolutional networks,” IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499-1503, 2016, doi: 10.1109/LSP.2016.2603342.

[11] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” arXiv preprint arXiv:1602.07261, 2016.

[12] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, pp. I-I, 2001, doi: 10.1109/CVPR.2001.990517.